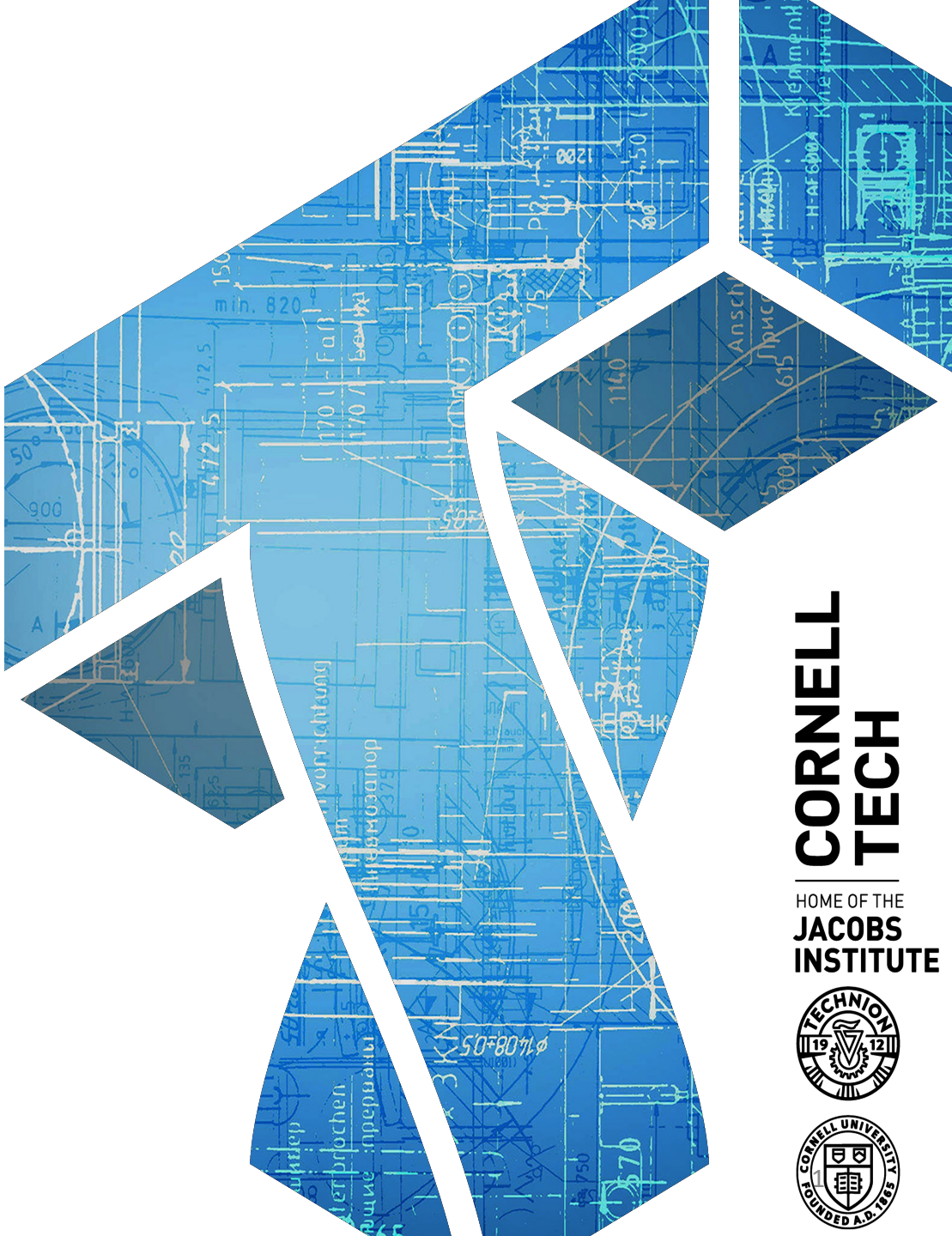
CS 5112 – 10/17
Network Flow Applications



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

- Simple s-t path: path from s to t with no repeated vertices
- s-t cut: partition (A,B) of V such that s in A and t in B
- Capacity of an s-t cut: c(A,B) = $\sum_{e \text{ out of } A} c_e$

Network Flow

Flow network is graph *G = (V,E)* with:

- edge capacities c_e
- designated nodes source *s* and sink *t*

An s-t flow is function $f : E \to \mathbb{R}^+$ such that:

I. (Capacity conditions) For each e in E, $0 \leq f(e) \leq c_e$

II. (Conservation conditions) For each node v besides s,t we have

$$f^{in}(v) = f^{out}(v)$$

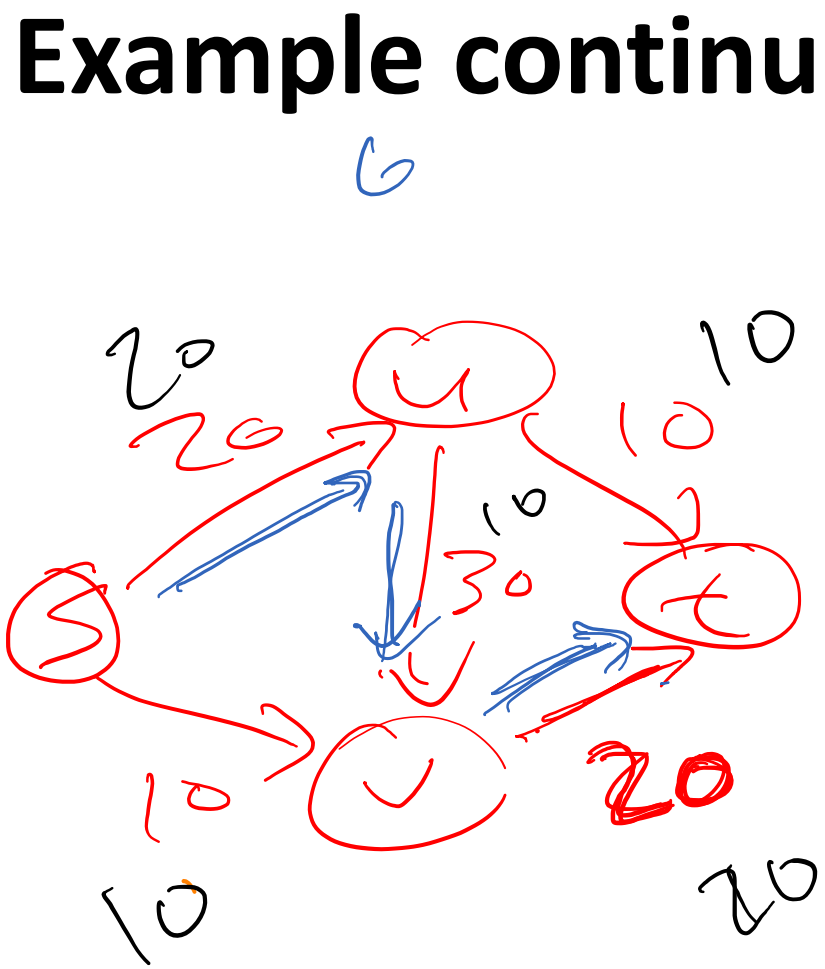
Example

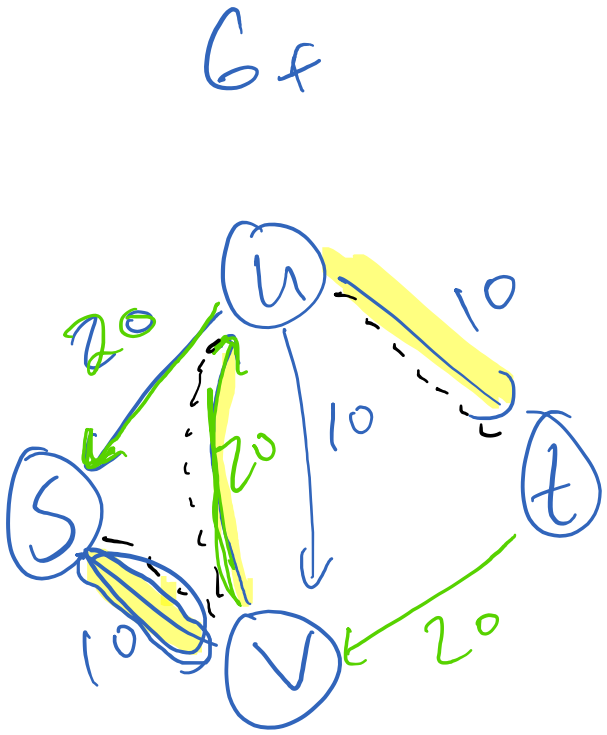
Ford-Fulkerson algorithm

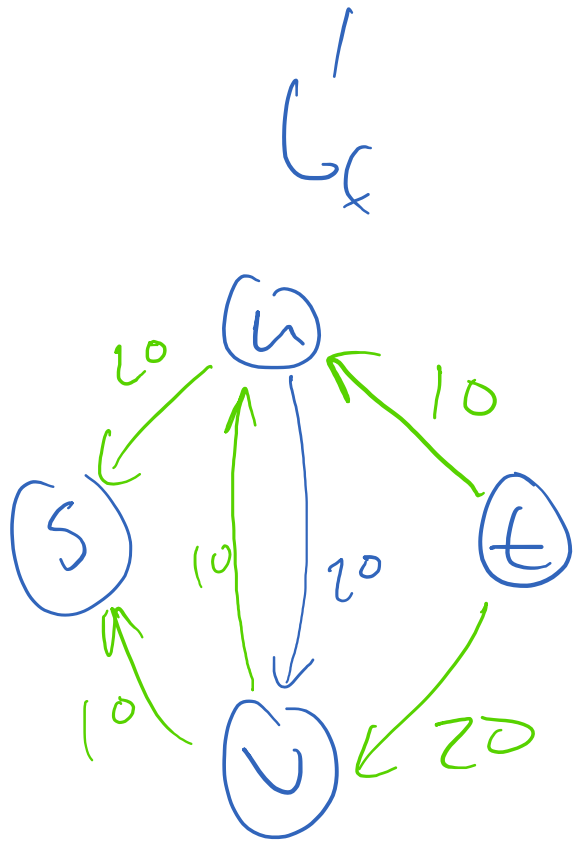
Ford-Fulkerson(G)
Foreach e in E : f(e) <- 0
Construct residual graph G_f
While exists s-t path P in G_f
    f <- Augment(f,c,P)
    Update G_f
Return f

Augment(f,c,P)
Let b = bottleneck(P,f)
For each (u,v) in P:
    If (u,v) is forward edge then
        increase f(e) by b
    else
        else decrease f(v,u) by b

Example continued







$f(s,u) = 20$

$f(u,v) = 20$

$f(v,t) = 20$

Ford-Fulkerson works

- ***Terminates***:
  - integer valued weights + flow always increases + upper bound on maximum flow
- ***Run time***: O(mC) where C is upper bound on maximum flow, and m term accounts for cost per iteration

Ford-Fulkerson works

- ***Correctness***:
  1. (7.8) For any s-t flow f we have that $v(f) \leq c(A,B)$ for any A,B cut
  2. (7.9) no s-t path in G_f => exists (A*,B*) cut with $c(A^*,B^*) = v(f)$
  3. FF terminates when no s-t path in G_f
  4. 1 + 2 + 3 => FF produces a maximum flow
- Analysis yields max-flow, min-cut theorem:

In every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut

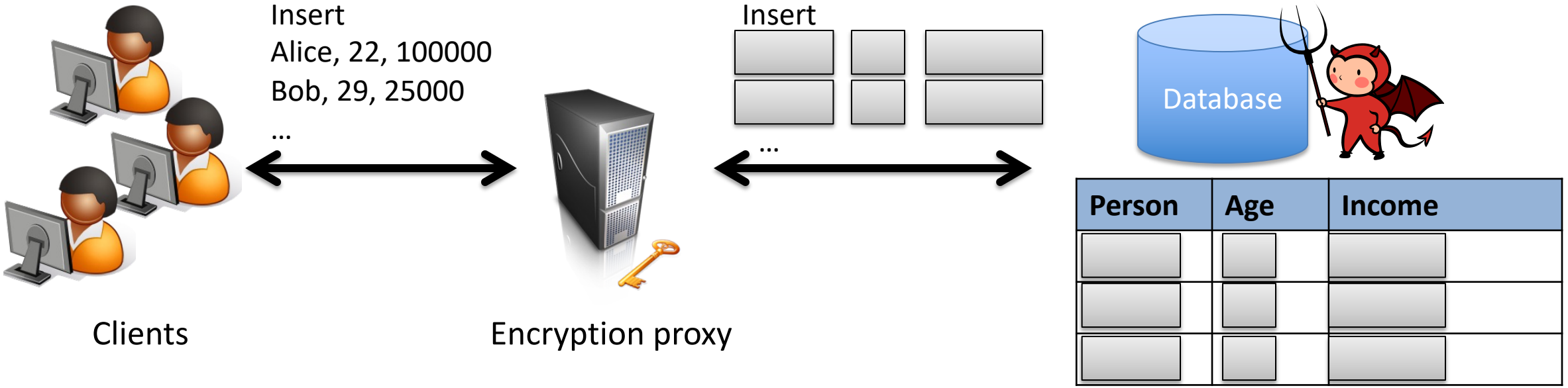
Why do we care about max flow, min cuts?

- Tons of problems can be modeled as flow problems!

Applications we'll cover

- Breaking database encryption
  - Bipartite matching
  - Minimum cost bipartite matching
- Survey design example
  - Circulations with demands and lower bounds

Database encryption



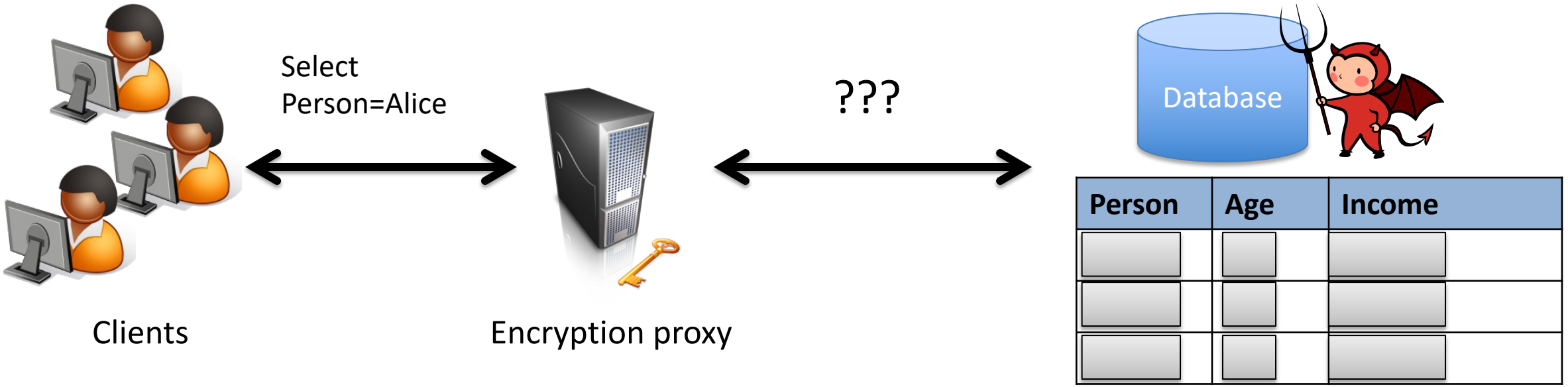
Encrypt data using secret key before insertion into database

Decrypt records fetched via queries

Encryption vs. DB functionality tension!
How can we fetch encrypted records?

Proxy can handle many clients (e.g., entire organization)

Database encryption



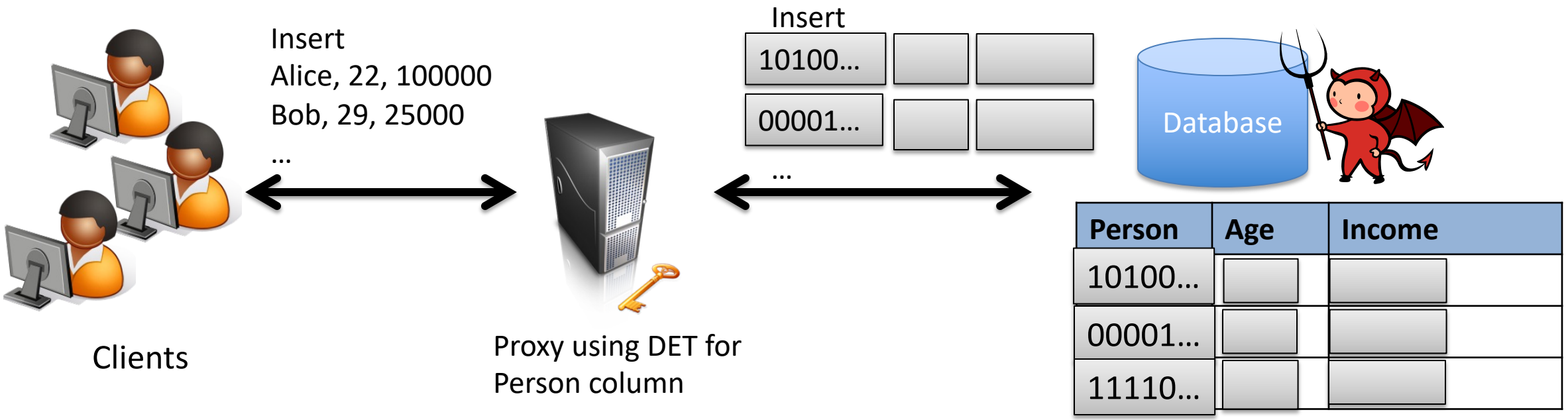
Encrypt data using secret key before insertion into database

Decrypt records fetched via queries

Encryption vs. DB functionality tension!
How can we fetch encrypted records?

Proxy can handle many clients (e.g., entire organization)

Using deterministic encryption (DET) in DBs



10100… = DET(Alice)

00001… = DET(Bob)

11110… = DET(Charlie)

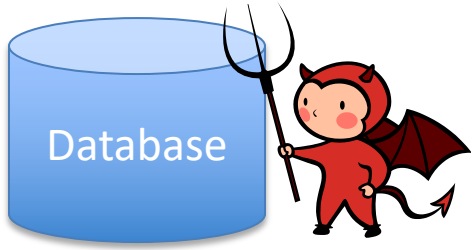
Using deterministic encryption (DET) in DBs

Clients

Select all rows with person = Alice

Proxy using DET for Person column

Select all rows with person = 10100…



Person	Age	Income
10100…		
00001…		
11110…		

Proxy recomputes C1 by encrypting search term Alice

10100… = DET(Alice)

Deterministic encryption enables fast retrieval of matching rows

CipherCloud

mongoDB.

baffle

PerspecSys
Making the Public Cloud Private

skyhigh

McAfee

bitglass

Microsoft

Leakage-Abuse Attacks against DET

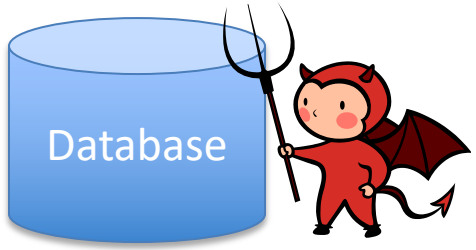
[Naveed, Kamara, Wright 2015]

Clients

Select all rows with person = Alice

Proxy using DET for Person column

Select all rows with person = 10100…



Person	Age	Income
10100…		
00001…		
11110…		
10100…		
00001…		
11110…		
11110…		
10100…		
00001…		
11110…		
11110…		

Adversary can generate histogram of ciphertexts

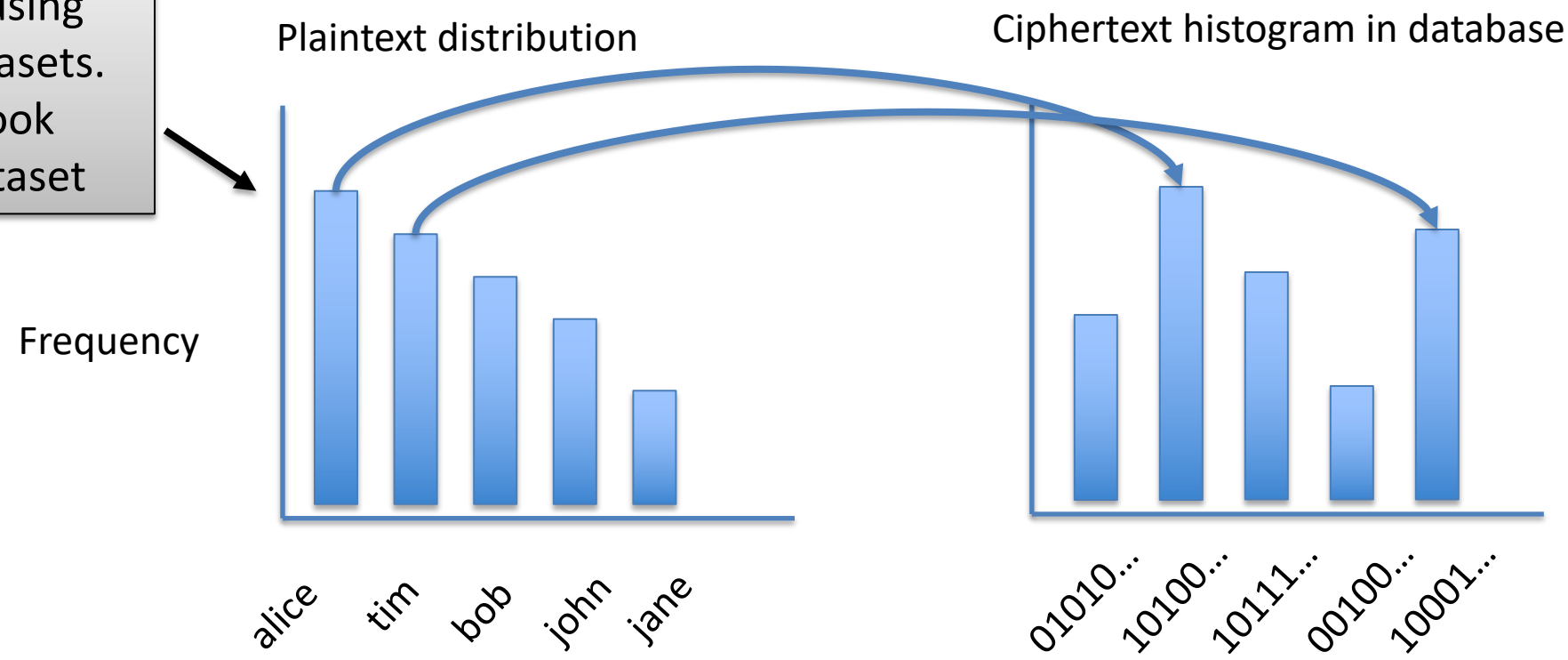
Given some info on plaintext distribution, use ***frequency analysis*** to recover plaintexts

Frequency attacks against DET

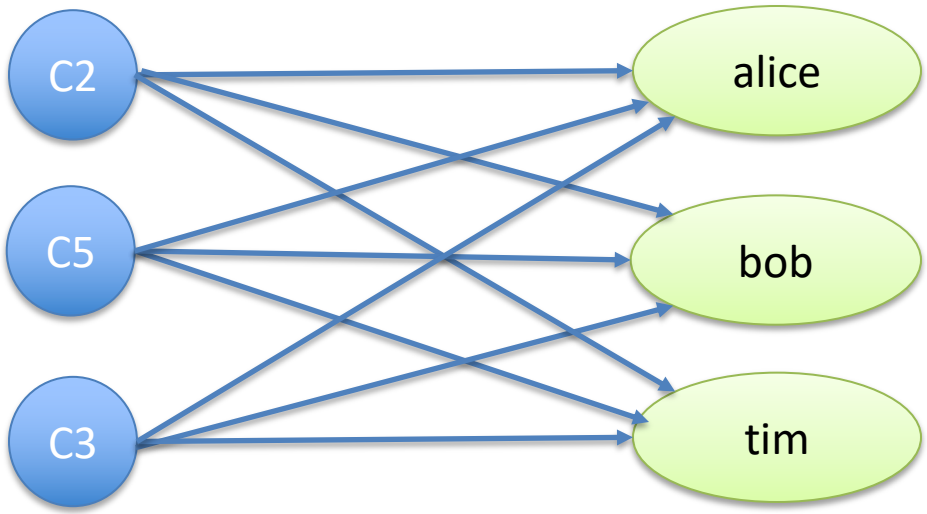
Idea: if attacker knows something about distribution of plaintexts, we can exploit it

[Al-Kindi 9th century BCE] [NKW 2016]

Estimate using public datasets. Ex: Facebook names dataset



Breaking deterministic encryption



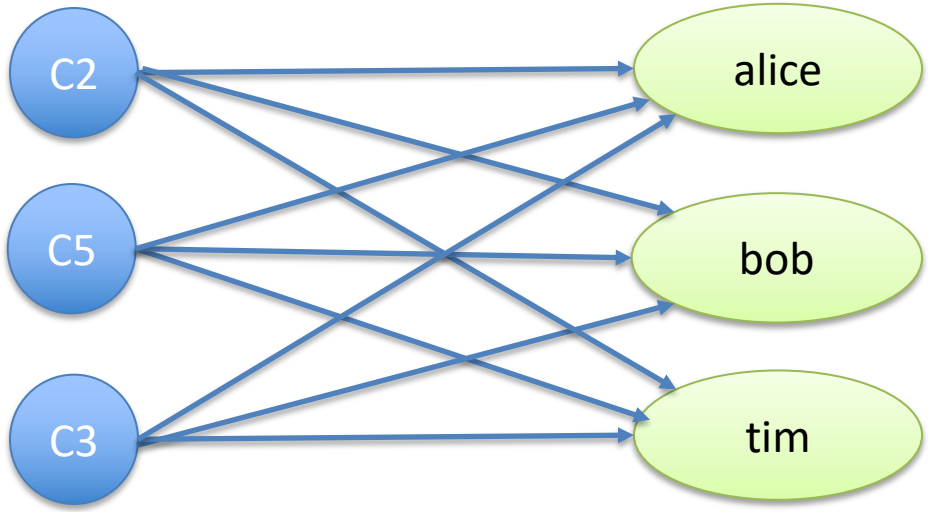
Bipartate graph problem: find matching of ciphertexts, names that minimizes total weight

$$\text{Weight}(C,\text{Name}) = |\text{Freq}(C) - \text{Freq}(\text{Name})|$$

Equivalent to frequency analysis

Bipartite matching

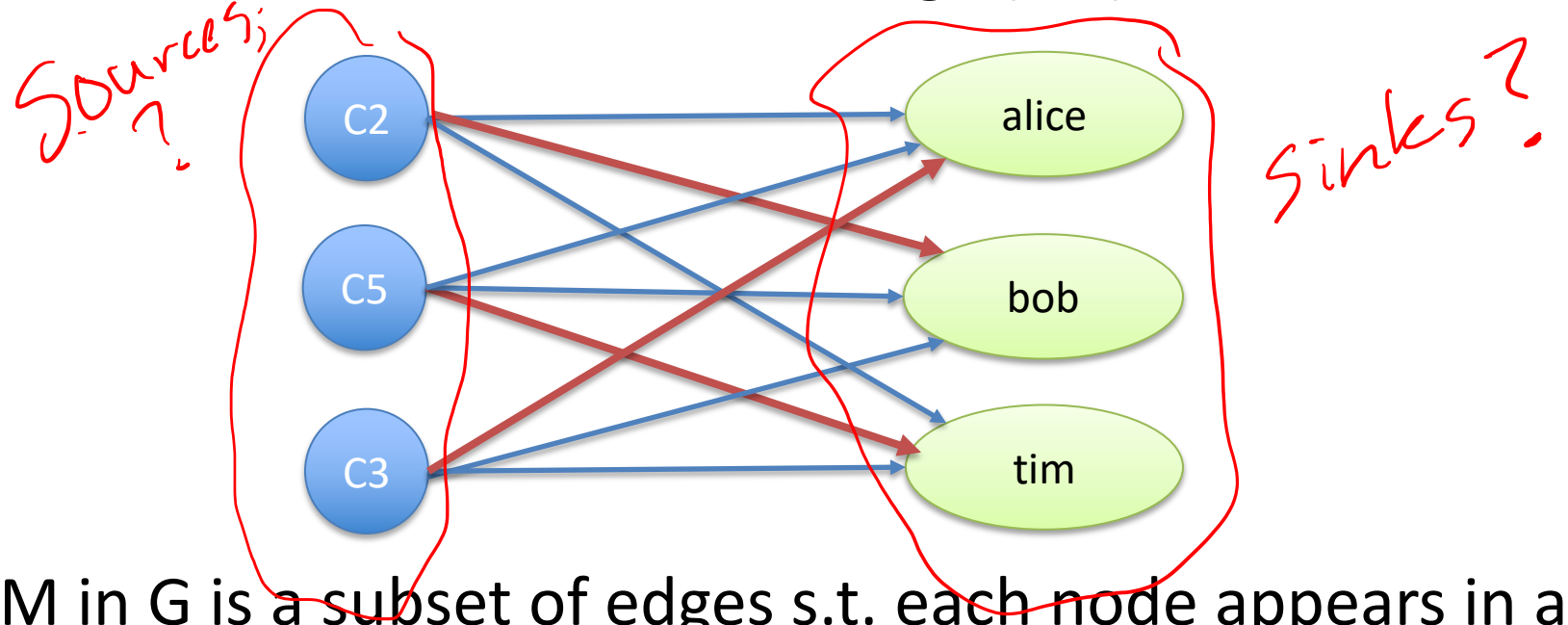
Bipartite graph G = (V,E) is an undirected graph whose node set can be partitioned as V = X ∪ Y such that ever edge (u,v) has u in X and v in Y



Matching M in G is a subset of edges s.t. each node appears in at most one edge in M

Bipartite matching

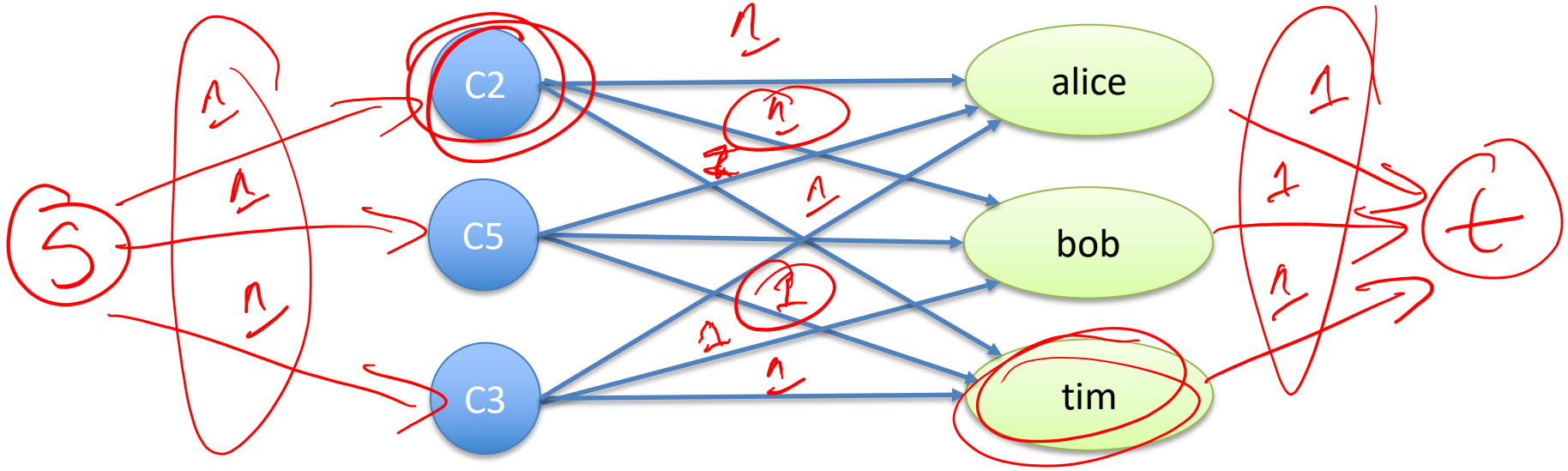
Bipartite graph G = (V,E) is an undirected graph whose node set can be partitioned as V = X ∪ Y such that ever edge (u,v) has u in X and v in Y



Matching M in G is a subset of edges s.t. each node appears in at most one edge in M

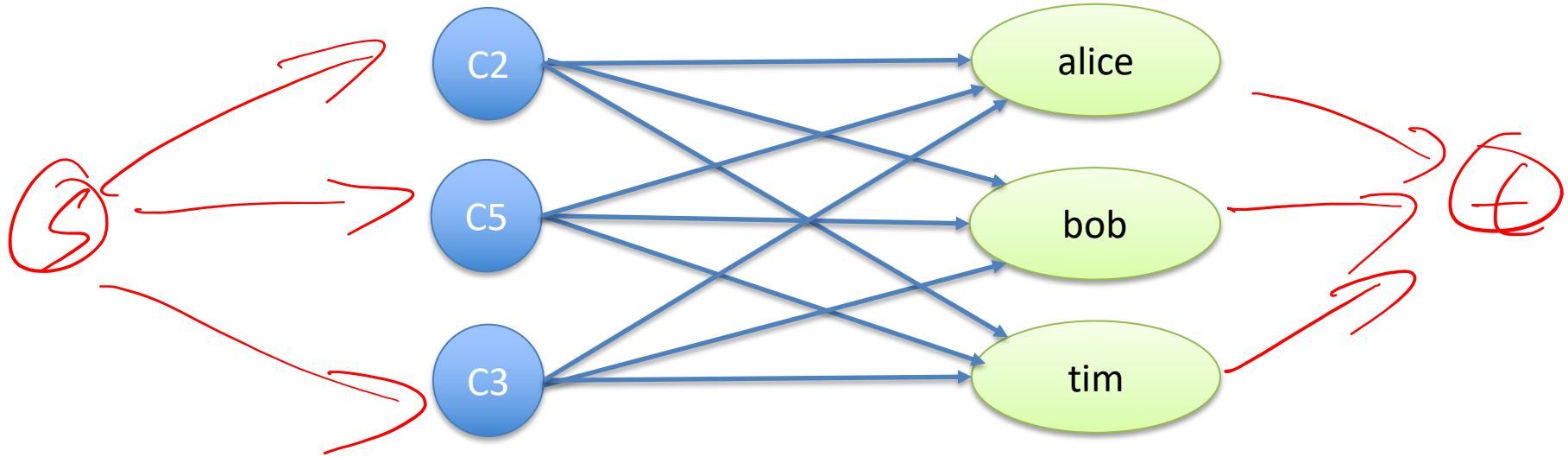
Bipartite matching problem: find matching with largest possible size

Bipartite matching using maximum flow



Bipartite matching using maximum flow

Create new graph G' with nodes s and t
Have edges from s to lefthandside nodes & edges to t from all righthandside nodes
All edges have capacity 1

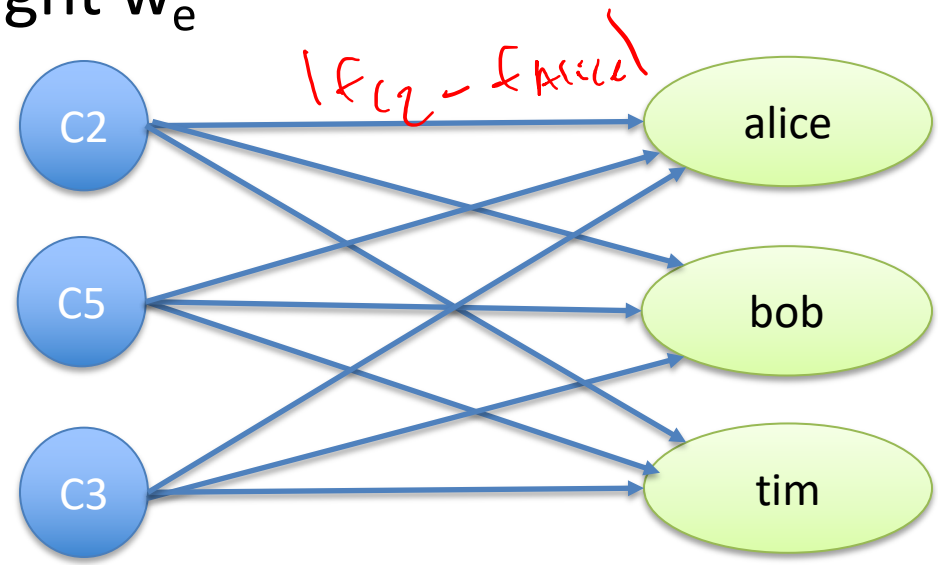


Ford-Fulkerson can be used to find maximum matching in O(mn) time

Minimum cost bipartite matching

Bipartite graph G = (V,E) is an undirected graph whose node set can be partitioned as V = X ∪ Y such that ever edge (u,v) has u in X and v in Y

Each edge has a weight w_e



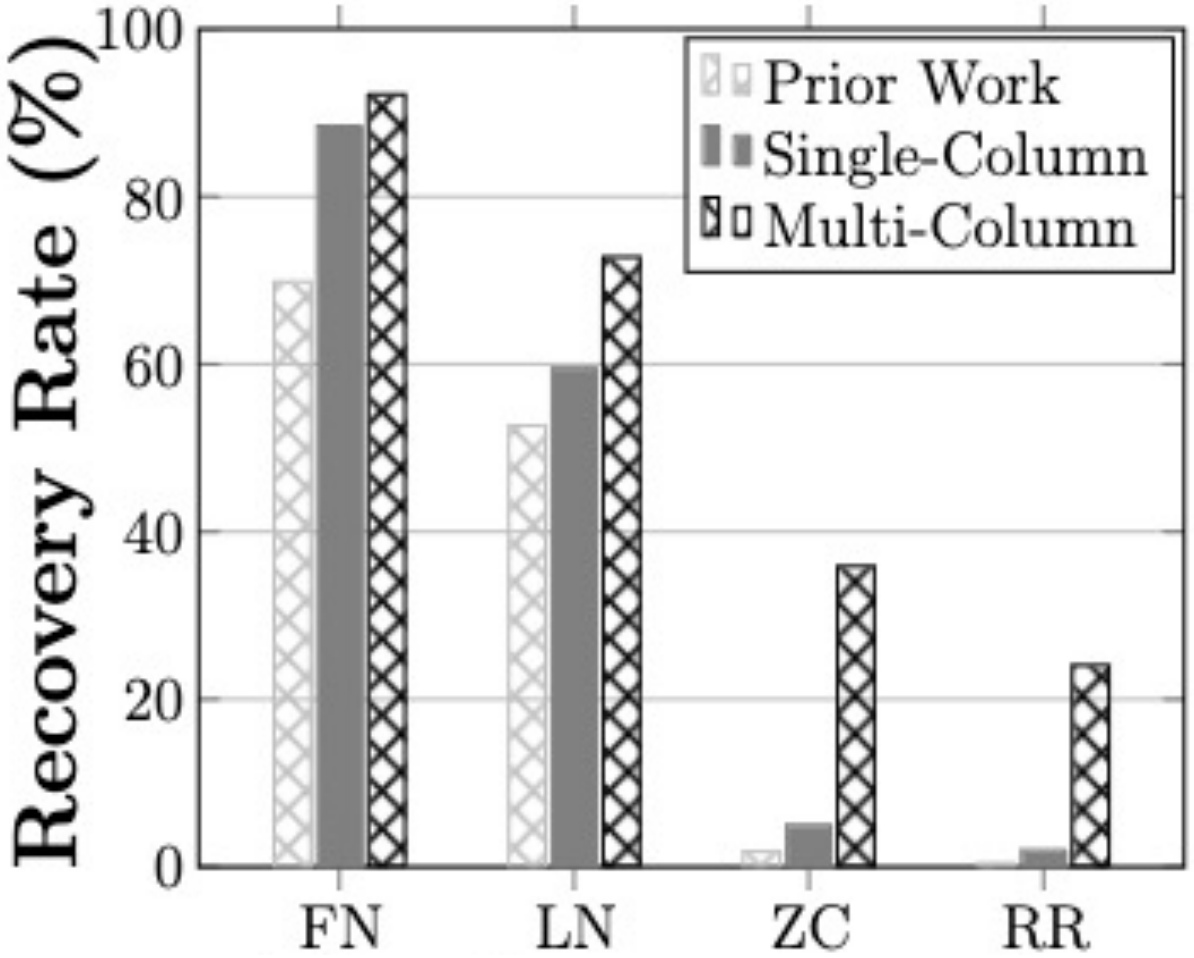
Matching M in G is a subset of edges s.t. each node appears in at most one edge in M. Cost(M) = $\sum_{e \in M} c_e$

Minimum cost bipartite matching: find perfect matching w/ min cost

Solving minimum cost bipartite matching

- Can we just use FF with edge capacities as weights?
  - This doesn't work
- Instead: modify our selection of residual s-t paths to take into account cost, then use similar structure as FF
- Called the Hungarian algorithm (see 7.13 for discussion of this problem)

Simulations of attacking encrypted database



Simulations of attacks using public datasets to recover encrypted first names (FN), last names (LN), zip codes (ZC), and all combined (RR)

[Bindschaedler et al. 2017]

Applications we'll cover

- Breaking database encryption
  - Bipartite matching
  - Minimum cost bipartite matching
- Survey design example
  - Circulations with demands and lower bounds

Survey design

Want to design survey for n customers

- Each customer receives questions about a subset of products
- Customer can only be asked about products they purchased
- Each customer asked about between c_i and c_i' products
- Each product j has responses from between p_j and p_j' customers

Circulation problems

- Extend maximum flow with multiple sources and sinks
- G = (V,E) with edge capacities and each node has demand d_v associated to it
  - $d_v > 0$ means v is a sink
  - $d_v < 0$ means v is a source
  - Let S be all sink nodes and T be all source nodes

A circulation is a function $f : E \to \mathbb{R}^+$ such that:

I. (Capacity conditions) For each e in E, $0 \leq f(e) \leq c_e$

II. (Conservation conditions) For each node v

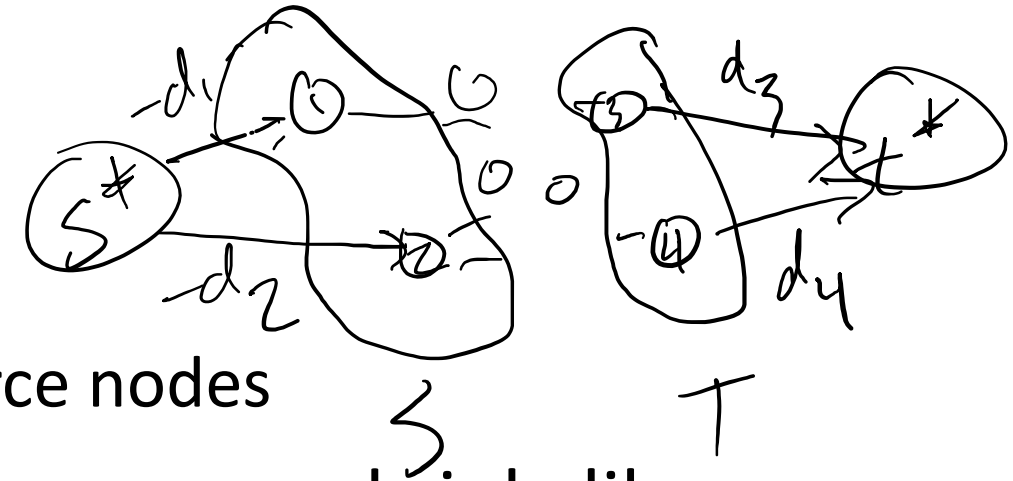
$$f^{in}(v) - f^{out}(v) = d_v$$

Circulation problems as max flow problems

- G = (V,E) with edge capacities and each node has demand d_v associated to it
  - $d_v > 0$ means v is a sink
  - $d_v < 0$ means v is a source
  - Let S be all sink nodes and T be all source nodes

Circulation problems as max flow problems

- G = (V,E) with edge capacities and each node has demand d_v associated to it
  - $d_v > 0$ means v is a sink
  - $d_v < 0$ means v is a source
  - Let S be all sink nodes and T be all source nodes



- Can simply add s* and t* as "super" source and sink, like we did for bipartite matching
  - Edge from s* to u in S has capacity $-d_u$
  - Edge from v in T to t* has capacity d_v
- Then max flow on new graph gives circulation

Circulation problems with lower bounds

- G = (V,E) with each node has demand d_v associated to it and each edge has
  - Capacity c_e' (same as before)
  - Lower bound $0 \leq c_e \leq c_e'$

A circulation is a function $f : E \to \mathbb{R}^+$ such that:

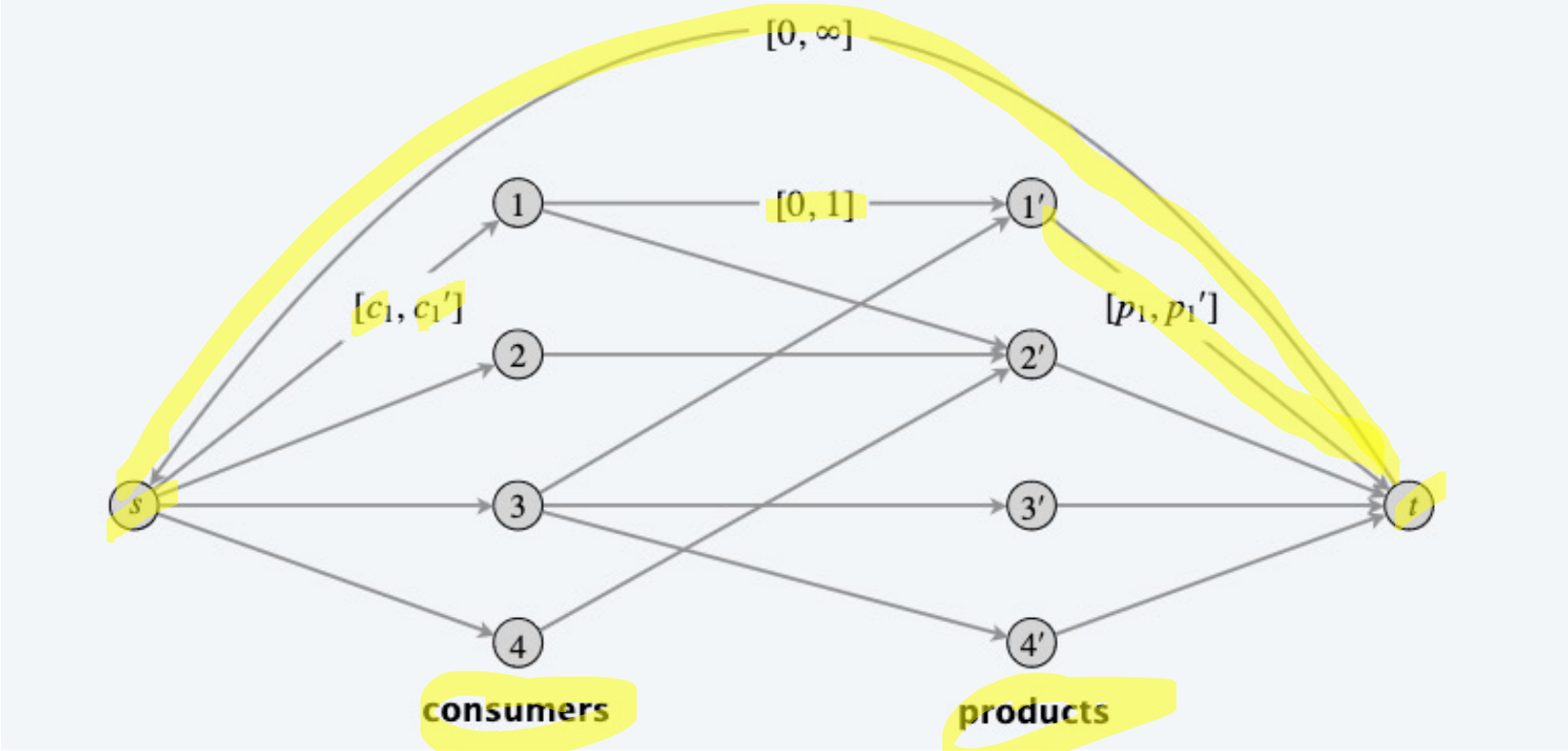
I. (Capacity conditions) For each e in E, $c_e \leq f(e) \leq c'_e$

II. (Conservation conditions) For each node v

$$f^{in}(v) - f^{out}(v) = d_v$$

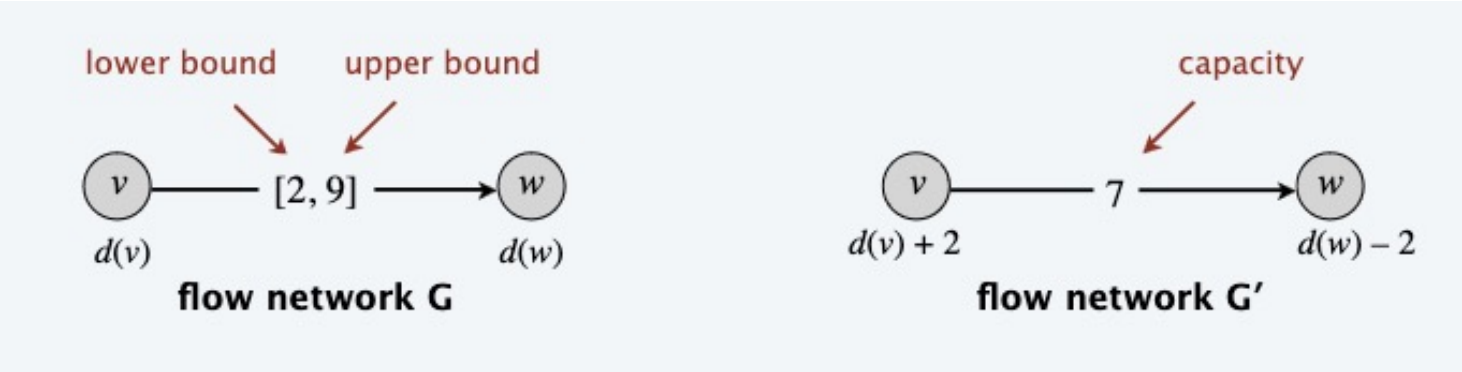
Survey design formalization

Bipartite G = (V,E) where V consists of customers and products
Edge between customer u and product v if u bought v
Edge from s to each customer with their constraints
Edge to t with product constraints
Edge from t to s



Circulation problems with lower bounds

- Reduce to circulation without lower bounds
  - Then we can use circulation approach which, in turn, uses max flow
- How?
  - Build a new graph G' that "embeds" the lower bounds into new graphs demands and capacities. Circulation solution on new graph gives circulation solution on original graph satisfying lower bounds



Network flow algorithms

- Powerful tool for building algorithms for a variety of tasks
- Come up in a variety of tasks very directly
- Clever ways to embed other problems into network flow problems, in order to solve them efficiently